A quoi sert un langage ?

Rôle d'un langage =

faciliter la mise en œuvre d'une machine complexe

permettre de transmettre un mode d'emploi pour réaliser une tâche avec une machine complexe

Définition qui amène trois questions :

- Qu'est-ce qu'une machine informatique ?
- Comment les définir et les construire ?
- Comment définir un langage pour une machine ?

Machine abstraite : définition

Machine abstraite =

Définition d'un mécanisme complexe par les commandes qu'il offre à l'extérieur

En informatique, une machine abstraite est généralement composée :

- d'un état
- des fonctions permettant de transformer et d'observer l'état

Les modules Caml de la bibliothèque de base sont des machines abstraites

Modèle de calcul : définition

Modèle de calcul =

Théorie mathématique qui permet de définir une classe de fonctions calculables ainsi que les règles permettant de déterminer le résultat d'un calcul.

En pratique, un modèle de calcul est construit en donnant :

- » la structure d'une machine abstraite
- » une technique de codage (représentation) des valeurs et des fonctions
- » un « moteur » qui enchaîne l'utilisation des règles

Concrètement, les ordinateurs sont issus de travaux sur les modèles de calculs

les langages de programmation aussi

Modèles

Les modèles de calculs sont innombrables !

- – La machine de Turing
  - » les ordinateurs modernes en sont une réalisation concrète (modèle de Von Neumann)
  - » C, Pascal, FORTRAN, etc. y ont leurs racines
- – Le λ-calcul (Church)
  - » Caml en est une déclinaison
- – La programmation logique (Robinson)
  - » Prolog, systèmes experts
- – Les réseaux de neurones
  - » un modèle exotique qui expliquera, peut-être, notre « ordinateur » intime ...

Caml (modèle simplifié)

Expression des fonctions

- le langage Caml au sens propre
  - mots-clés, règles d'écriture et de bonne formation

Machine abstraite

- un environnement
  - les liaisons (nom, valeur) à un moment donné
- l'expression courante
  - la dernière definition entrée

Moteur

- règles de typage
- règles d'évaluation
- "au ;;, typer l'expression courante puis l'évaluer puis modifier l'environnement si nécessaire"

μ-Logo

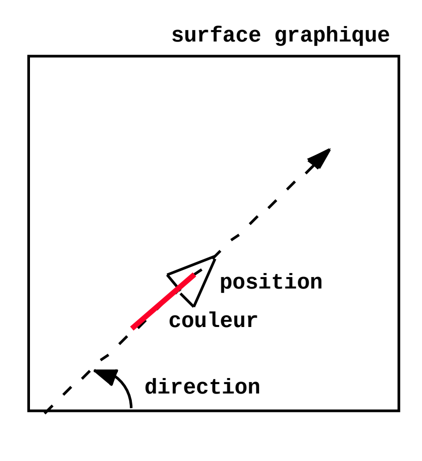
Objectif :

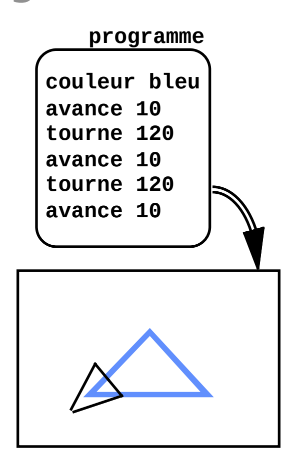
- construire un système complet de programmation
  - » définir un modèle de calcul
  - » réaliser la machine abstraite et le moteur
  - » définir un langage de programmation
  - » réaliser le passage de la définition à l'exécution

Cadre

- petit langage graphique, inspiré de Logo
  - Logo : créé par Papert, à usage des très jeunes enfants pour favoriser la structuration de connaissances (mathématiques en particulier)
- une seule ambition : mettre en évidence quelques notions de base de l'informatique

Modèle μ-Logo





Modèle μ-Logo

Machine abstraite élémentaire

- une feuille graphique
  - nous réutilisons le module graphique de Caml
  - quadrillage (maille entière) avec un point de tracé et une couleur
- une tortue
  - » objet à champs mutables
    - direction (un angle)
    - position (deux valeurs entière)
    - couleur du tracé
- un ensemble de commandes pour la tortue

Programme (procédure)

- une liste de commandes

Note : le résultat d'un calcul sera un graphique !

Réalisation de la machine

3 problèmes :

- implanter la notion de tortue
- représenter les programmes
- construire le « moteur »

organisation du travail : 2 modules

- la tortue
  - définition du type
  - définitions des actions
- le moteur
  - définition du codage des programmes
  - exécution de ces programmes

Module tortue (interface)

```
module type TortueInterface =
sig
type turtle

val move_turtle : turtle -> float -> unit
val turn_turtle :  turtle -> float -> unit
val set_turtle :
    turtle -> float -> float -> float -> Graphics.color ->
unit
val create_turtle : unit -> turtle
val color_turtle : turtle -> Graphics.color -> unit
end;;
```

Programmation de la tortue

Utilisation du module "graphics" de Caml

5 primitives de base :
open_graph, clear_graph (nécessaire)
lineto, moveto, set_color

Le détail sera vu en TD et TP

Codage des programmes

Nous pouvons maintenant programmer des tracés

```
let equi = function l ->
  let t = create_turtle ()
  in begin
     open_graph "";
     clear_graph ();
     color_turtle t red;
     move_turtle t l;
     turn_turtle t 120.;
     move_turtle t l;
     turn_turtle t 120.;
     move_turtle t l;
     turn_turtle t 120.
     end ;;
```

Codage des programmes

MAIS

- il faut connaître Caml !
- Il y a beaucoup de redondance
- on peut faire plus simple !

```
let equi = function l ->
       execute_programme [liste d'actions elementaires];;
```

où `execute_programme` se chargerait des problèmes propres aux graphiques Caml (`open_graph` par exemple)

Problème

comment définir les actions élémentaires ?

Codage des programmes

Solution :

associer un **code** à chaque action
avoir une fonction qui associe aux code les fonctions Caml
avoir une fonction qui parcourt la liste des codes

Définition en Caml

```
type actions =
   Turn of float
|  Move of float
|  Pen of color;;
```

Codage des programmes

```
let rec do_action =
    function tortue ->
      function
        Move l -> move_turtle tortue l
      | Turn a -> turn_turtle tortue a
      | Pen c  -> color_turtle tortue c;;
```


Codage des programmes

```
let run_it =
  function tortue ->
      function l_actions ->
        let rec repete = function
              [] -> ()
           |  act::suite -> begin
                            (do_action tortue act);
                            (repete suite)
                            end
         in repete l_actions;;

let execute_programme =
  function actions ->
    begin
    open_graph "";
    clear_graph ();
    let t = create_turtle ()
    in run_it t actions
    end;;
```

Est-ce tout ?

Nous pouvons maintenant écrire

```
let equi = function l ->
       execute_programme [Move l; Turn 120.; Move l;
                          Turn 120.; Move l; Turn 120];;
```

C'est un progrès mais :

– ce serait bien de pouvoir se passer totalement de Caml !
– il faudrait pouvoir définir la procédure "equi" directement en µ-logo

Ce sera le sujet de la suite :-)